Fault Sensitivity Analysis Meets Zero-Value Attack

Oliver Mischke, Amir Moradi, Tim Güneysu
Horst Görtz Institute for IT-Security
Ruhr-Universität Bochum
Bochum, Germany
E-mail: {mischke, moradi, gueneysu}@crypto.rub.de

***Abstract*—Previous works have shown that the combinatorial path delay of a cryptographic function, e.g., the AES S-box, depends on its input value. Since the relation between critical path delay and input value seems to be relatively random and highly dependent on the routing of the circuit, up to now only template or some collision attacks could reliably extract the used secret key of implementations not protected against fault attacks. Here we present a new attack which is based on the fact that, because of the zero-to-zero mapping of the AES S-box inversion circuit, the critical path when processing the zero input is notably shorter than for all other inputs. Applying the attack to an AES design protected by an state-of-the-art fault detection scheme, we are able to fully recover the secret key in less than eight hours. Note that we neither require a known key measurement step (template case) nor a high similarity between different S-box instances (collision case). The only information gathered from the device is whether a fault occurred when processing a chosen plaintext.**

***Keywords*-Fault Attack; Fault Sensitivity; Fault Collision; Zero Value; AES;**

I. Introduction

Modern cryptosystems have been carefully designed to withstand any type of mathematical cryptanalysis. However, with the advent of implementation attacks on their physical instances, a powerful new class of attacks on modern cryptosystems, namely side-channel and fault-injection attacks, has entered the stage. These attacks not only consider the input and output of a cipher, but also take intermediate information into account that can be gathered or altered using different physical channels at runtime.

Examples of the currently known side-channels are timing behavior [10], power consumption [11], and electromagnetic radiation [23]. If faults can be injected into certain computation stages of an algorithm, e.g., by power or clock glitches, laser or UV light, radiation, etc., there exist another powerful tool to break a cryptographic implementation. For instance, in Differential Fault Analysis (DFA) [2] the information gained from faulty ciphertexts can be used when the attacker is able to specifically manipulate a round of a cipher. Fault Sensitivity Analysis (FSA) [14], on the other hand, does not rely on extracting information from the faulty ciphertext but instead only requires the information whether a ciphertext byte is faulty or not.

The authors of the original FSA article reported that the critical path delay of an S-box design is dependent on the input processed by AND and OR gates in ASIC implementations. The method of choice to recover this information is the use of clock glitches whose duration are reduced in small steps until a fault occurs or becomes stable. For the PPRM S-box [21], which has a very symmetric structure of AND and XOR gates, it was shown that the critical path delay is correlated with the Hamming weight of the S-box input. However, for an S-box being implemented by an inversion circuit followed by an affine transformation, it could not be shown yet how to map the timing information of the faults to input values of the S-box.

Contribution:

In this work we provide the first practical evidence that a zero-value model, which was first discovered in the context of multiplicative masking in [4], also is applicable in the context of FSA on S-boxes implemented using composite field arithmetic. Because of the special case that a zero input to the inversion circuit of the AES S-box is directly mapped to the zero output, the critical path delay of this input is especially short. This was first assumed as possibility in [20] and later confirmed by simulation in [17].

More importantly, this article demonstrates how this weakness can be exploited even if the evaluation circuit is equipped with a state-of-the-art fault detection scheme, which usually renders FSA-like attacks infeasible since they rely on observing the fault rates for each byte individually. Our evaluation circuit can mimic different concurrent error detection (CED) schemes, and we will demonstrate a successful complete key recovery on the invariance-based CED scheme presented at DAC 2012 [5].

Indeed, the only information required from the circuit to successfully mount our attack is the binary information whether or not a fault occurred while processing chosen 128-bit plaintext inputs.

Outline:

The remaining article is organized as follows: In Section II we briefly summarize previous attack techniques which we will need for our attack and introduce common CED schemes. The next section describes our architecture that practically realizes the presented CED schemes for later evaluation. The zero-value timing vulnerability of the AES S-box and our attack methodology is stated in Section IV. The practical application of this attack to our target design

is showcased in Section V. Finally, Section VI concludes this work.

II. Preliminaries

In the following we briefly review the Fault Sensitivity Analysis technique first published in [14] and subsequently refined in later publications [12], [13], [20]. We also give a short overview on recent Concurrent Error Detection (CED) schemes.

A. Fault Sensitivity Analysis (FSA)

FSA aims at recovering the critical path delay of S-box inputs by observing faulty ciphertext bytes. The showcased method to create a fault in [14] is to inject a clock glitch into the last round of the AES computation for which no MixColumns transformation is performed. By slowly reducing the duration of the clock glitch, one can observe at each step when ciphertext bytes are returned faulty. This can be used to deduce the critical path delay of the corresponding S-boxes in the last round. Note that – in contrast to [2] – not the faulty ciphertext value itself is used, rather the information whether a specific byte is returned faulty given a certain glitch duration.

If a model can be identified to explain the extracted timing characteristic, it can be used to recover the secret key. For example, the structure of a PPRM-Sbox [21] is very symmetric and only consists of AND and XOR arrays to minimize power consumption and the amount of toggles at the S-box output. Since it could be shown that AND gates exhibit a small delay when switching to zero inputs, the critical path can be well approximated by a Hamming weight (HW) model.

Some improvements to FSA have been published in [20], where the linear difference between key bytes could be recovered by correlation collision attacks first presented in [18]. Both presented improvements rely on the similarity of different S-box instances in the targeted ASIC implementations, which is usually not the case in FPGA-based implementations. As one of the improvements shown in [20], the distribution of the faulty ciphertext bytes for two S-box instances are compared to extract the secret. However, if a fault detection scheme is considered in the implementation, these and other FSA attacks are infeasible since the required byte-wise fault information or faulty outputs are no longer available from the circuit.

B. Concurrent Error Detection (CED) Schemes

Over the last decades several proposals for CED schemes have been published. In [6] the authors gave a current overview of common techniques which are depicted in Fig. 1. They can be grouped into four categories: information redundancy, time redundancy, hardware redundancy, and hybrid redundancy.

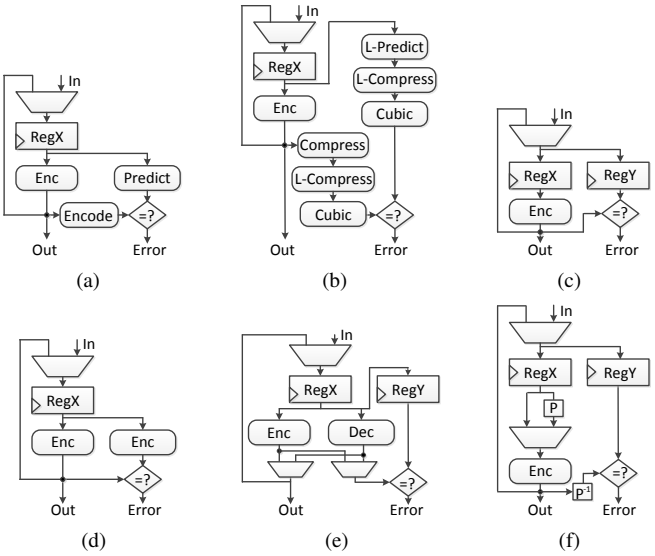


Figure 1. CED schemes: (a) information redundancy: parity; (b) information redundancy: robust code; (c) time redundancy; (d) hardware redundancy; (e) hybrid redundancy: inverse function; (f) hybrid redundancy: invariance-based CED (taken from [6]).

Information redundancy can for example be achieved by adding *parity* bits [9], [25] (Fig. 1(a)) or by a *robust code* (Fig. 1(b)) approach [7] which at the round input tries to predict a nonlinear property of the round output. **Time redundancy** [15] can be reached by computing the same cipher round twice using a single hardware instance in consecutive clock cycles (Fig. 1(c)). **Hardware redundancy** [15] on the other hand is achieved by duplicating the hardware instance and computing both the real cipher round and its check in the same clock cycle (Fig. 1(d)). The last category is **hybrid redundancy**, where different characteristics of the previous more general schemes are combined and sometimes rely on certain properties of the underlying algorithm.

For example, the scheme displayed in Fig. 1(e) is a variant of hardware redundancy, but instead of computing the same function twice, it inverts the output of the previous round and checks it against the input of that round [8], [24]. The newest scheme presented in [5] is the invariance-based CED (Fig. 1(f)) that is based on a variant of time redundancy. Again, instead of simply computing the same function twice and comparing both results, for the second execution the input and the output are additionally permuted exploiting internal properties of Advanced Encryption Standard (AES) [22].

III. Designing an Architecture for Evaluation

A major goal of our evaluation architecture is to test different countermeasures using exactly the same hardware instance. The invariance-based design of [5] has been reported to provide almost complete fault coverage under

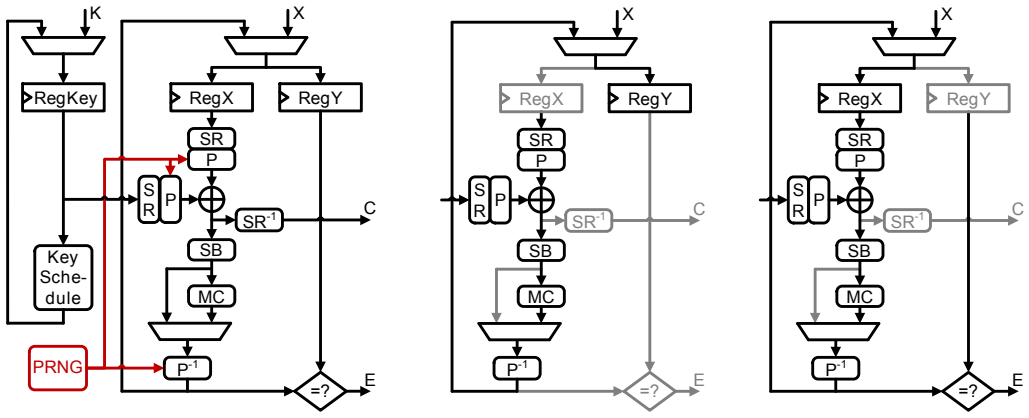


Figure 2. (a) Architecture of our evaluation circuit; (b) *computation step*; (c) *checking step*

different models, hence we adopted their design (that is depicted in Fig. 1(f)) as the base of our AES implementation. The evaluation architecture used in this work is depicted in Fig. 2.

The used abbreviations are: **SR** for **S**hift**R**ows, **SB** for **S**ub**B**ytes, and **MC** for **M**ix**C**olumns. **P** denotes a **P**ermutation, what is a slightly differently designed module compared to that proposed in [5].

As shown by Fig. 1(f), the original design contains a multiplexer to skip the permutation step. The permutation step itself was designed as a fixed column-based permutation. It is stated in [5] that only three permutations, denoted as P_1, P_2, and P_3, are possible if random inputs are given. Each round is computed in two clock cycles. In the first clock cycle, the plaintext saved in register **RegX** in a prior round, passes through the AES data path without entering the permutation and the result of this operation is stored in **RegY**. In the following we call this cycle the *computation step*. In the second clock cycle the data path including the permutation is used. Simultaneously with storing the computation result in **RegX** for the next round, the result is also compared to the previous result stored in **RegY** (denoted as *checking step* in the following). In case they differ, an exception is raised since a fault has been detected.

Note that for our design (Fig. 2) we slightly modify this approach by moving the ShiftRows operation to the beginning of the data path. Since SubBytes works solely on single bytes, **P** only operates on full columns and ShiftRows is simply a fixed wire permutation, this is possible without impact on area or performance. However, this simple but beneficial modification enables to implement **P** not as fixed permutation but as dynamic switch matrix which allows to swap columns in any way and differently in each clock cycle. The amount of possible valid permutations is thereby increased to $4! = 24$ obviously raising the complexity for an attack with respect to the original countermeasure as stated in [5].

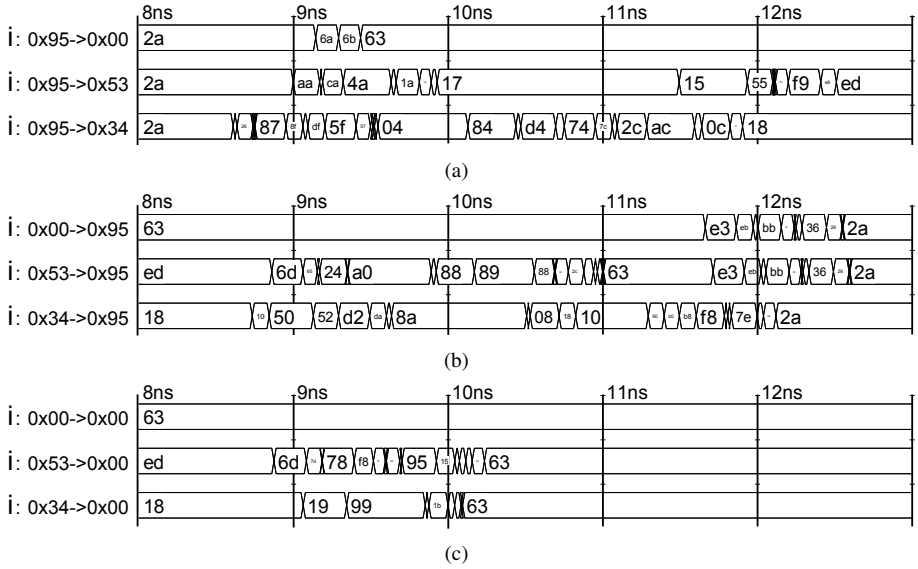
Since **P** can be adapted on-the-fly as part of the data path, we are able to imitate the behavior of several different CED schemes. If **P** is configured to operate as pass-through in both clock cycles of a round, this leads to a CED implementing *time redundancy*. If it operates as pass-through in the first clock cycle but performs a cyclic right shift of the columns in the second clock cycle, the scheme is identical to the *invariance-based hybrid redundancy* scheme of [5] (using the proposed permutation P_1). Finally, by randomly selecting the column order in both clock cycles we can implement a type of shuffling countermeasure which is sometimes applied to thwart side-channel attacks[1]. This also allows us to analyze possible side effects of combining fault and side-channel countermeasures.

Note that, if a different permutation for each clock cycle is selected and when looking at a single column, there exist only four different possibilities for which column will be processed by a certain hardware instance. This still significantly increases the difficulty for an attacker since there is only a 25% chance that the target column is processed by the target instance, while in the original scheme the column order for both the computation and checking step is fixed resulting in a 100% chance for the attacker.

IV. Zero-Value Fault Sensitivity Analysis

In the following we will briefly restate the input-dependent timing behavior of the AES S-box when implemented in combinatorial logic. Based on this, we show the zero-value vulnerability and describe the corresponding methodology to fully recover the secret key of our round-based AES design which is equipped with a state-of-the-art fault detection scheme and is described in Section III.

[1]Here, since our implementation realizes a round-based architecture and all S-boxes are performed at the same time, by *shuffling* we mean randomly assigning different S-box instances (resp. MixColumns) to the cipher state bytes.



(a)

(b)

(c)

Figure 3. Timing behavior of the used AES S-box [3]

A. Zero-Value Vulnerability

In [14] an input-dependent timing behavior of the AES S-box was first observed and exploited as Fault-Sensitivity Attack (FSA). Their results were confirmed in [19] by post place-and-route simulation. In the following, we review these findings by simulations based on the S-box of [3] as part of our architecture. The results shown here are based on the Virtex-5 LX50 FPGA of our target platform.

Figure 3(a) depicts the timing behavior until the S-box output becomes stable after a given input switches to three different exemplary values. First it confirms that the critical path delay is highly dependent on the new input given to the circuit. However, considering the results in Fig. 3(b) switching back and forth, it can be observed that not only the next state but also the previous (stable) state of the S-box influences the timing behavior of the circuit[2].

This shows that an attack model that takes both the next input and the previous state into account should be superior to one which only relies either on the next input *or* the previous state. Another important observation on the timing behavior is depicted by Fig. 3(c). It can be seen that the time required to have a stable output if the S-box input switches to the zero value is notably shorter compared to the other cases, which was also discovered in [20] and [17]. In fact, this is a variant of the zero-value vulnerability first described in [4]. In this work it was noticed that, when using multiplicative masking as countermeasure to thwart differential power analysis, the S-box circuit consumes notably less power when processing the zero input compared to all other cases since the zero value cannot be masked by multiplication. The same effect can be observed even if no masking countermeasure is applied to the AES S-box if the implementation is based on composite field arithmetic (as is the case in our designs). It is because during the inversion part of the S-box (at least) one operand of all multiplications are zero [16].

The special zero mapping of the S-box inversion circuit is an exception and respectively multiplications by zero are fast, hence the critical path is particularly short. This distinctive feature is obviously easier to exploit than the complex models based on the previous state and the next value. Therefore, we will now investigate the resistance of our CED design against an FSA-like attack using this zero-value attack model.

[2]It is, in fact, the same concept used in power analysis attacks when Hamming distance (HD) is applied to model the leakage associated to a sequential circuit.

B. Attack Methodology

Notations:

Let suppose that a clock glitch with length Δt is injected to the target device while it processes n plaintexts $X^{i\in\{1,\ldots,n\}} = \left(x_1^i, \ldots, x_{16}^i\right)$. Let also $f_{\Delta t}^i$ denote a random variable indicating whether a fault is detected while X^i was being processed. Now we define *total error rate* as

$$p_{\Delta t,\text{total}} = \frac{\sum_i f_{\Delta t}^i}{n}.$$

We also define *local error rate* associated to plaintext byte j as

$$p_{\Delta t,j}(x) = \frac{\sum_{i\in S_j^x} f_{\Delta t}^i}{|S_j^x|},$$

where S_j^x stands for a set of plaintexts whose j-th byte with $j \in \{1,\ldots,16\}$ is equal to x:

$$S_j^x \;=\; \{i \mid x_j^i = x\}.$$

For the sake of simplicity we omit the Δt term in later specification.

Attack Description:

Target of our attack is the first computation step in the first AES round. For clarity, we explicitely highlighted the data path in Fig. 2(b) in black. We assume that the critical path for each S-box is minimal whenever it processes a zero input, hence our final goal is to find the 16-byte plaintext X that force all S-box inputs to zero in the target clock cycle. While gradually shortening the injected clock glitch Δt, this byte configuration X will be the one which remains unaffected for the longest time. Since the S-box inputs of the first round are computed by xor'ing the plaintext bytes x_j with the corresponding key bytes k_j, this implies that the plaintext bytes must be equal to the key bytes: $\forall j,\ x_j = k_j$.

Note that it is not possible to directly measure the influence of each plaintext byte for a single computation i. If a fault is caused by the clock glitch during the computation on plaintext X^i, we can only deduce that at least one of the 16-byte values x_j^i caused the faulty execution, and is therefore not one of the candidates for the key bytes k_j. However, by repeating the experiments for a considerably large n, we can observe the local error rates p_j for each byte and discard those key candidates k_j which exhibit a significantly high local error rate ($p_j \gg p_{\text{total}}$). The rationale behind this decision is that, if a local error rate p_j is high compared to other values and therefore the corresponding k_j appeared in a higher amount of faulty X^i, it can be deduced that the critical path length of the candidate k_j is longer then the average and can therefore not be the correct candidate which should have ome of the shortest critical path lengths.

This is exactly the strategy that we follow as described in Algorithm 1. We can iteratively restrict the key space until only the correct key remains. The number of required iteration steps is influenced by the chosen threshold ϵ, which determines how many values are discarded from the set of remaining key candidates K_j (Algorithm 1, line 11). The smaller ϵ the more key candidates are discarded per iteration. Choosing a smaller ϵ also increases the chance to falsely exclude a correct key candidate k_j. In case the correct key candidate for a byte j has been excluded in a previous run, Algorithm 1 will later discard all other candidates in K_j and run Algorithm 2 to recover a valid set K_j as intermediate step (line 15).

While the situation is unlikely and was not observed during our evaluations, if for some reason a valid intermediate set K_j cannot be recovered, or the recovered full key after completion of Algorithm 1 is not the correct one, analog to a power analysis attack for each byte j the most likely candidates could be ranked according to the order they were excluded. This would allow an efficient brute force attack assuming that the number of uncertain sets K_j or their size is sufficiently small.

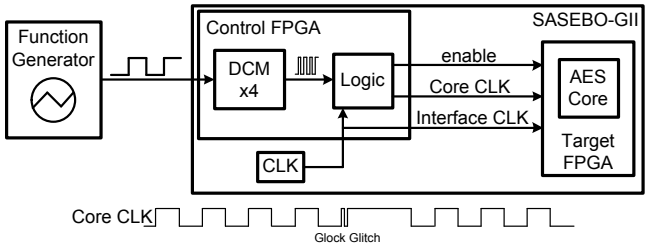


Figure 4. Block diagram of the experimental setup and timing diagram of the core clock

V. Evaluation of the Attack

In order to practically verify the feasibility of our attack we have chosen a SASEBO-GII [1] as evaluation platform. We implemented our fault-protected AES architecture in the Virtex-5 (XC5VLX50) FPGA on the evaluation platform, and injected faults from clock glitches generated by an Agilent 33521A Function Generator. Since this function generator only supplies signals with a maximum frequency of 30 MHz, we feed this clock signal to the control FPGA of the SASEBO-GII to increase its frequency using the clock multiplier of the digital clock manager (DCM). The control FPGA supplies the clock line to the target FPGA and can, at a desired clock cycle, create a clock glitch by switching between a slow clock and the faster clock originated from the function generator. The complete setup is depicted in Fig. 4.

In total we have analyzed thee different settings of the implemented CED architecture given in Section III, all requiring two clock cycles to compute a single AES round.

- In *profile* 1 we have configured the permutation matrix **P** to act as pass-through, i.e., no actual permutation is taking place in any clock cycle mimicking a simple time redundancy CED scheme. We use this design only for exploration purposes.
- During the evaluation of *profile* 2, which is the main focus of our work, the design acts equivalent to the invariance-based CED, where P_1 – defined in [5] – is used as the permutation matrix **P**.
- Finally, for *profile* 3 we configure **P** to randomize the order of the columns during every clock cycle of the AES computation, i.e., in both the *computation* and the *checking step*.

Note that the CED schemes implemented here all aim at detecting faults and, by subsequently zeroing the output, try to make it impossible for an attacker to use the faulty output in, e.g., differential fault analysis. When using an FSA-based technique, like in our case, we do not require the faulty output. Just the information that a fault occurred is sufficient to extract the secret. Also note that the CED schemes of *profile* 1 and *profile* 2 are practically equivalent during the *computation step*, which is the target of our fault injection. In other words, in all of the experiments shown

Algorithm 1: Fault-Sensitivity Analysis Exploiting Zero Values (FSA-ZV)

Output: Key candidate sets $K_{j\in\{1,\dots,16\}}$, forming 16-byte secret key(s)

Choose Δt to have a low total error rate ($0.1 < p_{\text{total}} < 0.2$)
for $j \in \{1,\dots,16\}$ **do**
  $K_j \leftarrow \forall k \in \{0,1\}^8$ /* initializing the sets */
end
while $\exists j, |K_j| > 1$ **do**
  Select $n = 25,000$ random plaintexts $X^{i\in\{1,\dots,n\}}$ as $x^i_j \in K_j$
  Send all n plaintexts to the target device while a clock glitch (Δt) is injected in the first computation of round 1
  Compute the local error rates $p_j(k),\ \forall j, \forall k \in K_j$
  $\mu = \frac{\sum_j \sum_{k\in K_j} p_j(k)}{\sum_j |K_j|}$ /* overall average */
  Find $\max = p_{j^*}(k^*)$ as $\forall j, \forall k \in K_j,\ p_{j^*}(k^*) \geq p_j(k)$ /* overall maximum */
  $\lambda = \mu + \epsilon; \quad \epsilon = \frac{\max - \mu}{3}$ /* threshold */
  for $j \in \{1,\dots,16\}$ **do**
    $K_j \leftarrow \{k \in K_j | p_j(k) < \lambda\}$ /* candidates with low error rate */
    if $K_j = \varnothing$ **then**
      $K_j \leftarrow$ **Algorithm 2**
    end
  end
  if $\mu < 0.1$ **then**
    Shorten Δt (nominally by 125ps)
  end
end

Algorithm 2: Recover Key Candidates

Input : $p_{j\in\{1,\dots,16\}}(k \in K_j)$, local error rates of candidates for the plaintext bytes j,
j^*, the index of the empty candidate list $K_{j^*} = \varnothing$,
λ, the threshold

Output: K_{j^*}, the new set of candidates for the plaintext byte j^*

for $\forall j \neq j^*$ **do**
  $\underline{k}_j = \underset{k}{\operatorname{argmin}}\ p_j(k)$ /* the most probable candidates */
end
Select $n = 25,000$ random plaintexts $X^{i\in\{1,\dots,n\}}$ as $x^i_{j^*} \in \{0,1\}^8$ and $\forall j \neq j^*, x^i_j = \underline{k}_j$
Send all n plaintexts to the target device while a clock glitch (Δt) is injected
Compute the local error rates $p_{j^*}(k),\ \forall k \in \{0,1\}^8$
$K_{j^*} \leftarrow \{k | p_{j^*}(k) < \lambda\}$

below we injected the faults by a clock glitch during the first *computation step*, i.e., when **RegY** stores the result of the first cipher round (see Fig. 2). Therefore, the *checking step* of all our design profiles have no impact on the efficiency of the performed attack.

A. Profile 1: Time redundancy-based CED

In order to practically verify the result achieved by simulation, e.g., Fig. 3, we performed some experiments on *profile* 1.

By setting all plaintext bytes x_j to the correct key bytes k_j we could confirm that we are still able to inject a fault, but by a very short Δt. By sending plaintexts X^i, where only one plaintext byte $x^i_{j^*}$ is random and the others are still equivalent to the correct keys ($x^i_{j\neq j^*} = k_j$), we are able to observe the local error rates p_{j^*} for all possible values x_{j^*}. Figure 5 shows the local error rates p_8 when shortening Δt from 17ns to 12ns in steps of 250ps.

One can see that the local error rates for each input usually rise from 0% to 100% over a Δt span of 1.5ns − 2ns and are distributed over the whole tested spectrum. While this allowed us to clearly discern the order in which each plaintext value gets faulty when shortening Δt for S-box 8, examining the same for other S-boxes showed dissimilarity

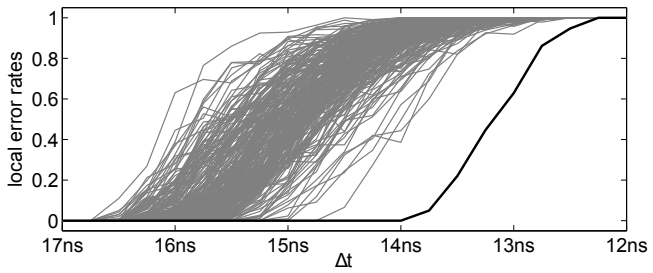


Figure 5. Evolution of local error rates of S-box 8 while shortening the clock glitch length on *profile* 1. The local error rate of the correct key candidate is highlighted in black.

between local error rates as well as critical path delay of different S-boxes. Because of routing differences the S-boxes are also too different from each other to mount e.g., a collision attack. However, more importantly, we could verify that when the S-box input equals zero, the faults can be induced only for a very short Δt.

B. Profile 2: Invariance-based CED

We now perform the complete key recovery attack described in Section IV on the invariance-based CED scheme. Note that, we do not force resets on any registers, therefore increasing the difficulty of the attack by accepting somewhat randomly precharged values on the registers during the attack execution.

Figure 6(a) shows the local error rates of all 16 S-boxes during the first exclusion round of Algorithm 1 where the key space has not been restricted yet. It can clearly be seen that the S-box computing on the x_{10}^i's has the highest impact on the observed local error rates. A $p_{10}(k)$ of 90% here means that 90% of X^i where $x_{10}^i = k$ were faulty and therefore candidate k is unlikely to be the correct key candidate k. Redoing the same experiment, without restricting the key space and computing the Pearson's correlation coefficient between the local error rates of each S-box (Fig. 6(b)), shows that only S-box 10 and – to a smaller degree – S-box 12 are having an impact on the observed local error rates. This is the reason why we exclude improbable key candidates based on the observed mean and maximum local error rates. A different straightforward choice would be to define a threshold ϵ (Algorithm 1, line 11) for key exclusion individually for each S-box. If so, we would falsely discard key candidates based on local error rates which were not caused by the critical path delay of this S-box but on others instead.

Note that the reason why only S-box 10 and S-box 12 are having an impact on the error rates initially is that their critical path is longer compared to other S-box instances and they therefore are affected by the shortened clock glitches first. Once the wrong key candidates corresponding to the longer critical path delays have been excluded the other S-boxes start to get affected by the faults as well.

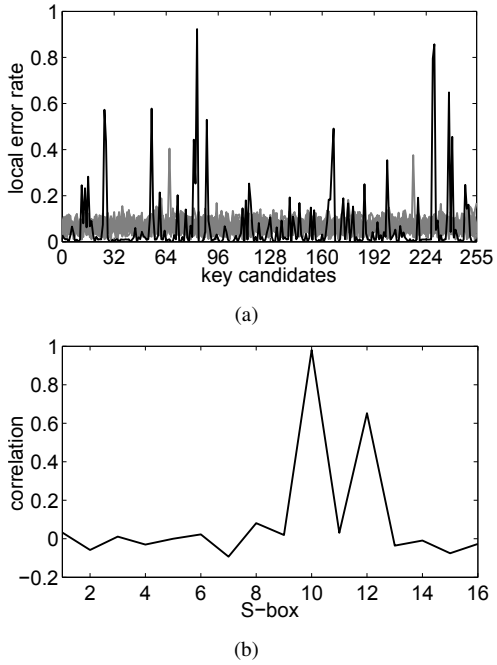


Figure 6. (a) Local error rates computed by the first round of Algorithm 1 on *profile* 2 with S-box 10 highlighted and (b) correlation between local error rates obtained by two runs of the first round of Algorithm 1 (with no prior excluded candidates)

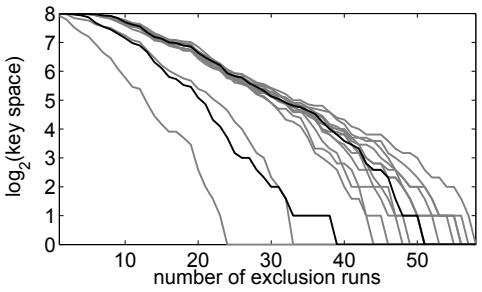
Figure 7 depicts the remaining key space after each exclusion round of Algorithm 1 both per S-box (Fig. 7(a)) and for the complete AES key (Fig. 7(b)). After 58 iterations of Algorithm 1, the complete secret could be extracted.

Each exclusion round was completed in about 8 minutes sending each plaintext X^i and receiving the response over the SASEBO-GII UART connection from the control FPGA. The time for a complete recovery could either be reduced by creating the X^i of each round on the control FPGA to reduce the communication bottleneck or by reducing n (Algorithm 1, line 6) for further iterations of the algorithm. The parameter n could be chosen based on the number of remaining key candidates per S-box.

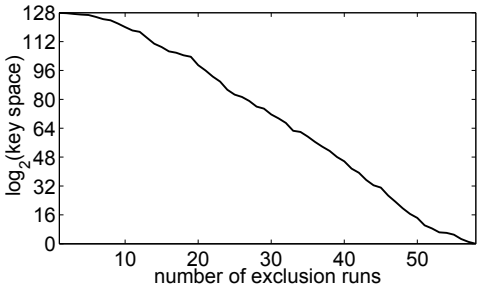
C. Profile 3: Randomized permutations

When randomizing the position of the columns during every clock cycle, this obviously complicates the attack. Now each plaintext byte will randomly be processed by one of four possible S-box instances (since the row index of each byte remains constant). Since the local error rates are no longer dependent on a specific S-box instantiation but are potentially influenced by four instances, it is harder to exclude keys since if k_j has a high local error rate in one S-box instance, this could be mitigated by having lower local error rates in other instances.

Because of this randomization we do not see different timing characteristics for each S-box. The observed behavior of each row of the cipher state will basically be the same



(a)



(b)

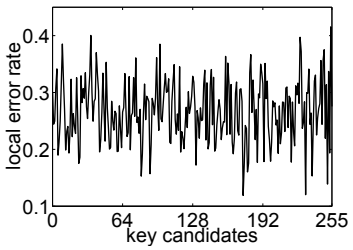
Figure 7. Remaining key space after each exclusion round; (a) per S-box, (b) for the complete 128-bit key (*profile* 2)

since the used S-box instances in each row of the state are randomized for every computation. By iterating Algorithm 1 we can still discard unlikely key candidates, and once the candidates with the highest local error rates are excluded, we can skip to Algorithm 2 to retrieve local error rates for each S-box.

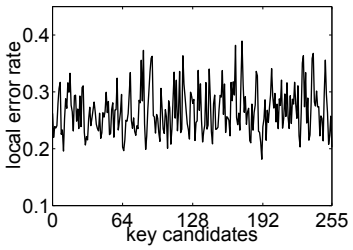
Figure 8(a) depicts the local error rates p_6 gathered by performing Algorithm 2 after the remaining key space K_j has been reduced to 32 per S-box on average. The same is shown for p_{10} in Fig. 8(b). While we cannot directly recover the corresponding key bytes from these measurements, it is possible to recover the linear difference between the correct key bytes. We used the same technique as [18], but on local error rates instead of mean power traces. In this setting in order to recover the correct $\Delta k = k_6 \oplus k_{10}$, correlation between $p_6(\forall k \in \{0,1\}^8)$ and $p_{10}(k \oplus \Delta k)$ is estimated. The result which is shown by Fig. 8(c) indicates the efficiency of the scheme. This way we can recover all Δk of each row, reducing the remaining key space to 2^8 per row and to 2^{32} in total.

VI. Conclusions

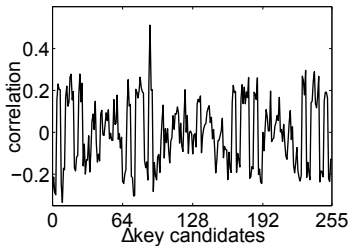
We have presented a case study of a fault-protected AES encryption engine where S-boxes are realized as combinatorial circuits. We exploit the fact that the critical path delay is dependent on their input which is especially short in case it is the zero value. Contrary to some previous work based on collisions, we do not require that the S-box instances exhibit a high similarity to each other. We were able to extract the secret key from the circuit based on the concept of an



(a)



(b)



(c)

Figure 8. (a), (b) local error rates obtained by running Algorithm 2 on *profile* 3 for (a) S-box 6 and (b) S-box 10; (c) correlation collision attack recovering the correct $\Delta k = k_6 \oplus k_{10}$

enhanced fault sensitivity analysis, even though the applied CED scheme usually renders FSA-like attacks infeasible.

The attack works whenever the implemented circuit performs a normal AES encryption round followed by checking operation in a different clock cycle. This is for example the case if an *invariance-based* CED [5] is used or one utilizing *time-redundancy*. The time required to completely recover the key of our design is less than eight hours. In our case the attack speed could further be increased significantly by improving the measurement setup, since the currently used slow UART connection to request each AES execution is the bottleneck of the attack flow. This would, however, not be the case if the actual computation is the bottleneck, as can be the case if the target is a smart card. We have not verified the attack against other CED schemes like those computing the inverse of a previous round in parallel. However, previous work [19] has shown that some information can be recovered if the targeted part of the circuit is *on* the critical path and is thereby affected by the attack first.

We have also shown that in case a countermeasure like column shuffling is implemented to thwart fault or power analysis attacks, this might even increase the susceptibility

of the implementation to successful attacks. Since the error rates of S-boxes processing the same row are unified and averaged by randomization, collision attacks become easily possible – although they were not feasible before due to the different S-box layouts. However, if the evaluation circuit would be equipped with a side-channel countermeasure such as boolean masking, this would very likely prevent an attacker from performing the proposed fault attack since the zero value vulnerability would effectively be negated by most masking schemes. This again demonstrates that countermeasures against different attacks should not be regarded independently of each other and potential interactions should be carefully analyzed.

ACKNOWLEDGMENT

This work has been partially funded by the European Union, Investing in your future, European Regional Development Fund, and the German Federal Ministry of Economics and Technology (Grant 01ME12025 SecMobil).

REFERENCES

[1] Side-channel Attack Standard Evaluation Board (SASEBO). Further information are available via http://www.risec.aist.go.jp/project/sasebo/ and http://www.morita-tech.co.jp/SAKURA/en/hardware.html.

[2] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems. In *CRYPTO 1997*, volume 1294 of *LNCS*, pages 513–525. Springer, 1997.

[3] D. Canright. A Very Compact S-Box for AES. In *CHES 2005*, volume 3659 of *LNCS*, pages 441–455. Springer, 2005. The HDL specification is available at the author's official webpage http://faculty.nps.edu/drcanrig/pub/index.html.

[4] J. D. Golic and C. Tymen. Multiplicative Masking and Power Analysis of AES. In *CHES 2002*, volume 2523 of *LNCS*, pages 198–212. Springer, 2002.

[5] X. Guo and R. Karri. Invariance-Based Concurrent Error Detection for Advanced Encryption Standard. In *DAC 2012*, pages 573–578. ACM, 2012.

[6] X. Guo, D. Mukhopadhyay, and R. Karri. Provably Secure Concurrent Error Detection Against Differential Fault Analysis. Cryptology ePrint Archive, Report 2012/552, 2012. http://eprint.iacr.org/2012/552.

[7] M. G. Karpovsky, K. J. Kulikowski, and A. Taubin. Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard. In *DSN*, pages 93–101. IEEE Computer Society, 2004.

[8] R. Karri, K. Wu, P. Mishra, and Y. Kim. Concurrent Error Detection Schemes for Fault-Based Side-Channel Cryptanalysis of Symmetric Block Ciphers. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 21(12):1509–1517, 2002.

[9] M. M. Kermani and A. Reyhani-Masoleh. Concurrent Structure-Independent Fault Detection Schemes for the Advanced Encryption Standard. *IEEE Trans. Computers*, 59(5):608–622, 2010.

[10] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO 1996*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.

[11] P. C. Kocher, J. Jaffe, and B. Jun. Differential Power Analysis. In *CRYPTO 1999*, volume 1666 of *LNCS*, pages 388–397. Springer, 1999.

[12] Y. Li, K. Ohta, and K. Sakiyama. An Extension of Fault Sensitivity Analysis Based on Clockwise Collision. In *Inscrypt*, volume 7763 of *LNCS*, pages 46–59. Springer, 2012.

[13] Y. Li, K. Ohta, and K. Sakiyama. New Fault-Based Side-Channel Attack Using Fault Sensitivity. *IEEE Transactions on Information Forensics and Security*, 7(1):88–97, 2012.

[14] Y. Li, K. Sakiyama, S. Gomisawa, T. Fukunaga, J. Takahashi, and K. Ohta. Fault Sensitivity Analysis. In *CHES 2010*, volume 6225 of *LNCS*, pages 320–334. Springer, 2010.

[15] T. G. Malkin, F.-X. Standaert, and M. Yung. A Comparative Cost/Security Analysis of Fault Attack Countermeasures. In *FDTC*, pages 159–172. Springer, 2006.

[16] S. Mangard, E. Oswald, and T. Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, 2007.

[17] F. Melzani and A. Palomba. Enhancing Fault Sensitivity Analysis Through Templates. In *HOST*, pages 25–28. IEEE, 2013.

[18] A. Moradi, O. Mischke, and T. Eisenbarth. Correlation-Enhanced Power Analysis Collision Attack. In *CHES 2010*, volume 6225 of *LNCS*, pages 125–139. Springer, 2010.

[19] A. Moradi, O. Mischke, and C. Paar. One Attack to Rule Them All: Collision Timing Attack versus 42 AES ASIC Cores. *IEEE Trans. Computers*, 62(9):1786–1798, 2013.

[20] A. Moradi, O. Mischke, C. Paar, Y. Li, K. Ohta, and K. Sakiyama. On the Power of Fault Sensitivity Analysis and Collision Side-Channel Attacks in a Combined Setting. In *CHES*, volume 6917 of *LNCS*, pages 292–311. Springer, 2011.

[21] S. Morioka and A. Satoh. An Optimized S-Box Circuit Architecture for Low Power AES Design. In *CHES*, volume 2523 of *LNCS*, pages 172–186. Springer, 2002.

[22] National Institute of Standards and Technology (NIST). *Announcing the Advanced Encryption Standard (AES)*. Nov. 2001. Published: Federal Information Processing Standards Publication 197.

[23] J.-J. Quisquater and D. Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In *E-smart 2001*, volume 2140 of *LNCS*, pages 200–210. Springer, 2001.

[24] A. Satoh, T. Sugawara, N. Homma, and T. Aoki. High-Performance Concurrent Error Detection Scheme for AES Hardware. In *CHES 2008*, volume 5154 of *LNCS*, pages 100–112. Springer, 2008.

[25] K. Wu, R. Karri, G. Kuznetsov, and M. Gössel. Low Cost Concurrent Error Detection for the Advanced Encryption Standard. In *ITC*, pages 1242–1248. IEEE, 2004.